CENTRE DE RECHERCHE
SUR LES TRANSPORTS
(C.R.T.)
CENTRE FOR RESEARCH
ON TRANSPORTATION

UNIVERSITÉ DE MONTRÉAL
ÉCOLE DES HAUTES ÉTUDES COMMERCIALES
ÉCOLE POLYTECHNIQUE

CENTRE DE RECHERCHE SUR LES TRANSPORTS
UNIVERSITÉ DE MONTRÉAL
C.P. 6128, SUCCURSALE A,
MONTRÉAL
CANADA H3C 3J7

TÉLÉPHONE: (514) 343-7575
TÉLEX: 05-24146 BIBPOLYTEC
EMAIL: crt@crt.umontreal.ca
TÉLÉCOPIEUR (FAX): (514) 343-7121

CRT-675

SOLVING THE MAXIMUM CLIQUE PROBLEM USING A TABU SEARCH APPROACH*

by

Michel Gendreau
Louis Salvail
Patrick Soriano

* The authors are grateful to the Quebec Government (Fonds F.C.A.R.) and to the Canadian Natural Sciences and Engineering Research Council (grant OGP0038816) for their financial support.

Centre de recherche sur les transports - Publication #675

December 1989

90-03008/

Dépôt légal - 1er trimestre 1990

ABSTRACT

We describe two variants of a Tabu Search heuristic, a deterministic one and a probabilistic one, for the maximum clique problem. This heuristic may be viewed as a natural alternate implementation of tabu search for this problem when compared to existing ones.

We also present a new random graph generator, the $\tilde{p}$-generator, which produces graphs with larger clique sizes than comparable ones obtained by classical random graph generating techniques.

Computational results on a large set of test problems randomly generated with this new generator are reported and compared with those of other approximate methods.

Key words : Maximum Clique, Tabu Search, Probabilistic Tabu, Random Graph Generator, Approximate Methods.

- - - - - - - -

RÉSUMÉ

Nous décrivons deux variantes, l'une déterministe et l'autre probabiliste, d'un heuristique de recherche avec tabous pour le problème de clique maximale. En regard des autres méthodes de recherche avec tabous pour ce problème, notre approche représente une implantation alternative naturelle.

Nous présentons aussi un nouveau générateur de graphes aléatoires, le générateur $\tilde{p}$, qui produit des graphes avec des cliques de plus grande taille que les graphes comparables obtenus par des méthodes de génération de graphes aléatoires classiques.

Nous donnons des résultats numériques pour un ensemble important de problèmes aléatoires et nous les comparons à ceux d'autres méthodes de résolution approchée.

Mots-clés : clique maximale, recherche avec tabous, tabou probabiliste, génération de graphes aléatoires, méthodes de résolution approchée.

1. Introduction

Given a simple undirected graph G=(V,E), a **complete subgraph** is one whose vertices are all pairwise adjacent. A **clique** is a maximal (inclusionwise) complete subgraph and a **maximum clique** is a clique whose cardinality is maximum.

The problem of finding a maximum clique in G is equivalent to the problem of finding a **maximum vertex packing** (independent or stable set of vertices) or a **minimum vertex cover** (set of vertices covering-meeting-all edges) in the complement of G (the graph $\bar{G}=(V,\bar{E})$ where $\bar{E}$ is the complement relative to VxV of E).

These three problems are important since they occur in a variety of applications: information retrieval systems [8], signal transmission analysis [3, 34], classification theory [7], sociological structures [23, 33], economy [1, 36], timetabling and many others. In timetabling for instance, the size of the maximum clique in the conflict graph of a set of activities is a lower bound on the number of periods required to schedule these activities, while the size of a maximum vertex packing is an upper bound on the number of activities that can be scheduled simultaneously.

With regards to computational complexity, the maximum clique problem (MCP) and its equivalents have been shown to be NP-hard many years ago (see Garey and Johnson [12]). However, there are a number of special classes of graphs for which polynomial algorithms exist (see [21] for a discussion of those). As for arbitrary graphs, a number of non-polynomial algorithms mostly of the branch-and-bound or enumerative type, have been proposed [2, 6, 9, 14, 17, 26, 28, 29, 32, 35], as well as a few heuristic or approximate methods [13, 25].

In the last ten years or so, Glover [18, 19] and independently Hansen and Jaumard [22] have developed a general metaheuristic for tackling difficult combinatorial optimization problems.

This method which is known as **Tabu Search** (TS), or Steepest Ascent/Mildest Descent in Hansen and Jaumard's terminology, combines a local search procedure with a number of clever anti-cycling rules which prevent the search from getting "trapped" in local optima. It has been successfully applied to the Traveling Salesman Problem [19], graph coloring [24], neural networks [37] and flow shop sequencing [38] among others.

Recently, Friden et al. [10, 11] have developed a TS heuristic for finding stable sets in large graphs (STABULUS) and a TS based exact algorithm for determining the maximum independent set in a graph (TABARIS). Both of these methods have displayed impressive computational results.

In this paper, we describe two variants (a deterministic one and a probabilistic one) of a TS heuristic for the MCP, which may be viewed as a natural alternate implementation of TS for this problem when compared to STABULUS. We also present a new random graph generator, the **$\hat{p}$-generator** which produces test problems with larger clique size than comparable problems obtained by classical random graph generation techniques.

The paper is organized as follows. The basic principles of TS are briefly recalled in section 2. In section 3, we describe in detail our implementation of deterministic TS for the MCP. The probabilistic variant of this algorithm is presented in section 4. Section 5 is devoted to a short discussion of the main differences between our approach and STABULUS. In section 6, computational results on a large set of randomly generated test problems are reported and compared with those of other approximate methods (greedy heuristics, simulated annealing-type methods). The $\hat{p}$-generator is also described in that section. Section 7 concludes the paper.

2. Basic principles of tabu search

In this section, we only briefly sketch the basic elements of the Tabu Search metaheuristic. A full description of the method can be found in the fundamental papers of Glover [18, 19] or in the tutorial article by de Werra and Hertz [37]. It should be noted that our presentation closely parallels those of Friden et al. in [10] and [11] and that mostly the same notation is used.

Given a function f to be maximized over some set X (normally finite), TS starts from some initial feasible point in X and proceeds iteratively from one point in X to another until some termination criterion is met. At that time, one hopes that the optimal solution, or at least a very good approximate solution, has been found.

To each solution s in X, one associates a **set of neighbours** $N(s) \subset X$ which can be readily obtained from s. The basic iterative step then then consists in randomly generating a sample N*(s) of neighbours of s (note that in many applications, it may be legitimate to take N*(s)=N(s)) and then moving to the best solution s* in N*(s) (i.e., $s^* \in \underset{s' \in N^*(s)}{\operatorname{argmax}} \{f(s')\}$).

It should be stressed that since there is no guaranteed strict improvement in the value of the objective function from one iteration to the next, such a procedure may very well cycle. To prevent this from happening, in TS one or several **tabu lists** are introduced. These lists are used to record historical information on the path in X followed by the search procedure during the previous iterations. This information is then used to exclude moves which would tend to make the search process go back to a previously visited solution. Classical examples of tabu lists are: a list of the k last solutions examined (where k is fixed or variable) or one of

the k last modifications (changes which occur when we move from a solution s to the next one s*) that were made (in this case we forbid the reverse modification from occurring).

When one uses a tabu list of modifications, rather than previous solutions, it may occur that some "good" solutions may not be considered at some iteration (because the move from the current solution to these is tabu). It may thus be desirable in certain conditions to cancel the tabu status of a move. One way of accomplishing this is via an **aspiration level** mechanism. This concept is extensively discussed in de Werra and Hertz [37], but for our purposes an illustrative example will suffice. Consider a problem in which the objective function f (to be maximized) may only take integer values. To each possible value z of f, we assign at the start the aspiration level A(z)=z+1. In the course of the search, whenever we consider going from a solution s to a solution s' by a tabu move, we cancel the tabu status of the move if $f(s') > A(f(s))$. The aspiration level of f(s) is updated to take the value f(s')+1 after any move from s to s' such that $f(s') > A(f(s))$. It is easily seen that even though we allow for the cancellation of the tabu status of some moves, this will not by itself induce cycling since the move from s to s' can only be executed once under tabu status.

We mentioned earlier that the search procedure will keep on going until some termination criterion is met. In most applications, this criterion is simply that the search will stop when a specified number of iterations have been performed without finding a better solution than the best solution currently at hand. When the optimal value of the objective function is known beforehand or has been determined during the search process (in some cases, this may be accomplished through some bounding procedures), the search may obviously be discontinued as soon as a solution with that objective value is found.

3. Deterministic tabu search for the maximum clique selection

In this section, we present a variant of our TS heuristic for the MCP in which no probabilistic elements come into play. In particular, we do not use sampling in the neighbourhoods of solution points.

Basic framework

In many combinatorial optimization problems, the first critical decision which must be addressed when applying TS, is the choice of the solution set to be explored by the search. In the context of the MCP, one should first notice that exploring the set of cliques would be totally impractical, since cliques by themselves are difficult to identify. On the other hand, it is easily seen from basic definitions that a maximum clique is simply a complete subgraph of maximum cardinality. Furthermore, a complete subgraph can be completely described by its vertex-set. If one denotes by X the set of vertex-sets of complete subgraphs in G, then finding the maximum clique in G amounts to solving the optimization problem $\max_{S \in X} f(S) = |S|$.

The set X also lends itself to a nice neighbourhood structure. To better describe it, we first introduce some necessary notation. For $v \in V$, let B(v) be the set of vertices that are adjacent to v in G, i.e., $B(v) = \{v' \in V \mid (v, v') \in E\}$, and for $S \in X$ denote by C(S) the set of vertices that are adjacent to all vertices in S, i.e., $C(S) = \bigcap_{v \in S} B(v)$. For $S \in X$, also define $N^-(S) = \{S' \subset V \mid S' = S \setminus \{v\},\ v \in S\}$ and $N^+(S) = \{S' \subset V \mid S' = S \cup \{v\},\ v \in C(S)\}$. $N^-(S)$ is then the set of all complete subgraphs which may be obtained from S by deleting a single vertex in it, and $N^+(S)$ is the set of complete sub-

graphs which may be obtained by adding to S a single vertex which is adjacent to all elements of S. By setting $N(S)=N^-(S) \cup N^+(S)$, we thus end up with a consistent neighbourhood structure on X. Also note that $S' \in N^-(S) \iff S \in N^+(S)$, which implies that this neighbourhood structure is symmetric.

It is interesting to remark that this neighbourhood structure has implicitly been exploited by many branch-and-bound algorithms for the MCP (see for instance [14, 32]). We have also used it successfully in a parallel project [16] dealing with the application to the MCP of the simulated annealing method introduced by Kirkpatrick et al. [27].

Another nice feature of this neighbourghhood structure is that, when dealing with augmenting moves, i.e. moves from a solution S to a neighbour S' in $N^+(S)$, the neighbourhood to be considered next is easily determined due to the fact that $C(S')=C(S) \cap B(v')$, where $\{v'\}=S' \setminus S$. When dealing with decreasing moves, i.e. to an S' in $N^-(S)$, however, one must recompute C(S') from scratch.

At this point, we should carefully analyze the basic step in the iterative procedure. Given a current solution S, one notices that $f(S')=|S|+1$, $\forall S' \in N^+(S)$, and $f(S')=|S|-1$, $\forall S' \in N^-(S)$. Two implications are to be drawn from this: (1) we should always move to an S' in $N^+(S)$ whenever possible; (2) the objective function gives us no information regarding the specific S' in $N^+(S)$ (or in $N^-(S)$ when no move in $N^+(S)$ is possible) which should be selected. One could arbitrarily select any S', but such an approach cannot be expected to be successful since it amounts to letting the search wander somewhat aimlessly. A better way of dealing with this problem is to add a secondary criterion to break ties. In our implementation, we have used a **greedy selection rule**: we move to the neighbour for which $|C(S')|$ is maximal. Such a move is the one which has the greatest

potential for producing a large clique, since $|S'| + |C(S')|$ is an upper bound on the size of any clique containing S'. When it is applied to $N^+(S)$, this rule is equivalent to the one proposed by Johnson [25] in his greedy heuristic for the MCP, since it amounts to choosing the vertex v' of largest degree in the restricted subgraph $G(S)=(C(S), E(S))$, where $E(S)=\{(v, w) \in E | v \in C(S), w \in C(S)\}$ (see also [14, 15] for further details).

A direct consequence of this choice of tie-breaking rule is that, since we do not use sampling, if we start the search procedure with $S=\emptyset$ (which we do), it will proceed directly to the clique that would be obtained by the application of Johnson's [25] greedy heuristic. This will provide us early with a fairly good solution and sometimes with an optimal one (see [15, 32] for extensive discussions on the quality of the solutions produced by this heuristic).

Tabu lists

In our implementation, we have experimented with two tabu lists. The first one T_1 is simply a list of the last $|T_1|$ solutions visited, while the second T_2 is a list of the last $|T_2|$ vertices that were **deleted.** $|T_1|$ and $|T_2|$ are fixed parameters that are specified as inputs to the procedure.

The list T_1 is used at all times (i.e. whether S' is in $N^+(S)$ or $N^-(S)$), while T_2 is used only when making augmenting moves. The reasons for this unusual scheme are two-fold: (1) it was possible using clever programming tricks to maintain a list of previously visited solutions at reasonable cost; since this allowed for interesting experimentation with the procedure, it was judged to be a valid addition; (2) the set of

complete subgraphs is a fairly constrained search space in which the addition of a vertex to the current solution will in many cases orient the search in a drastic way; conversely, removal of a vertex will often "open up" the search; for these reasons, it was decided not to restrict deletions, but only additions. The relative merits of each of these lists will be discussed and analyzed in section 6.

Aspiration level mechanism

With regards to our two tabu list scheme, it should first be noted that no aspiration level mechanism can (and for that matter should) allow for the cancellation of moves which are tabu in T_1, since it is a list of previously visited solutions.

As for moves which are tabu relative to T_2, we may wish to cancel their tabu status using the simple aspiration level function described in the previous section. For our specific application, this mechanism can be effectively implemented in an implicit form by adding a test which will allow us to move to a solution S' if $f(S') > z^*$ where z^* is the size of the largest clique found so far.

To see why this test is the only one required to implement this, consider the following facts:

- since at the start of the procedure one has $z^*=0$, and since the objective function at most increases by 1 in any iteration, if the current value of z^* is k, this implies that z^* must have taken all the integer values in the range [0, k] in previous iterations; one must therefore have visited solutions for which an improvement was found for every value of f between 0 and (k-1), which implies that $A(z)=z+2$, $z=0, 1, \ldots, k-1$;
- in an augmenting move, one always has $f(S')=f(S)+1$, which implies that $f(S') < A(f(S))$, unless $f(S)=k=z^*$, i.e., unless $f(S') > z^*$.

Re-directing search

As stated earlier, given any S' in X, $|S'| + |C(S')|$ is an upper bound on the size of a clique containing S'. If during the search process, we end up at a solution S such that $|S'| + |C(S')| < z^*$, $\forall S' \in N^+(S) \setminus T_1$ this means that there is no hope of finding a better clique through any non-tabu augmenting move, it is therefore advantageous to immediately redirect the search to some other portion of the solution space. If there exists non-tabu solutions in $N^-(S)$, we choose the best among these according to our tie-breaking rule, otherwise we arbitrarily choose any element in $N^-(S)$ to continue the search from.

Termination criterion

We terminate the procedure when MaxIter iterations without improvement in z* occur. MaxIter is an input parameter to the procedure.

4. The probabilistic variant

We have also experimented with a probabilistic variant of the previous procedure. It differs from its deterministic counterpart by the following features:

(1) If a current solution S is a local maximum (i.e., if $N^+(S)=\emptyset$), the procedure randomly removes from S $\max\{k, |S|\}$ vertices where k is an input parameter; the search then proceeds from this new solution S'.

(2) If $N^+(S) \neq \emptyset$, a random sample N*(S) of size $\max\{L, |N^+(S)|\}$, where L is also an input parameter, is generated; if N*(S) contains at least one element S' such that moving to S' is non-tabu or S' allows for improvement of z*, the search proceeds to the element of N*(S) satisfying these conditions which is best according to the tie-breaking rule. Otherwise, a sequence of deleting moves is performed as in (1) above.

The second feature of this variant is simply the introduction of sampling for augmenting moves as described in section 2 above. The other feature which more closely resembles a "random shake-up" of the solution than anything else, has been motivated by our specific application: as mentioned earlier, computing C(S') after a deletion is much more expensive since it must be done from scratch; it is therefore advantageous to perform multiple deletions at the same time to cut down on these computations. We must mention however that preliminary computational results have indicated that finding the correct value of L for a given problem size is a tricky affair.

5. Main differences with STABULUS [10]

As mentioned in the introduction, STABULUS is a TS heuristic for finding stable (independent) sets in large graphs. It thus solves a problem which is equivalent to that of finding cliques.

The first notable difference between STABULUS and our method is that it is not strictly speaking an optimizing procedure: STABULUS will search a graph for a stable set of size k with k being a specified input parameter. If one wants to use STABULUS to find a maximum independent set, he first has to "guess" the size of that set and then search the graph for such a set. It is well-known that for certain types of graphs, this value can be readily estimated, but one may wonder about the performance of STABULUS on "unknown" graphs.

The second striking difference between STABULUS and our approach is with regards to the solution set which is explored: in STABULUS, the authors have taken advantage of the fact that they know the size of the stable set that they are searching for; the solution set explored is that of subsets of k vertices. Such a choice of solution set implies that their procedure progresses from a subset which is "almost" an independent set to another, until a stable set (the equivalent of a complete subgraph in

clique terminology) is found, at which time the procedure can be immediately terminated since the desired solution has been found. One may thus say that STABULUS is a "dual-type" procedure, while ours which moves in the space of feasible solutions (complete subgraphs), represents a primal approach.

6. Computational results

To test our procedure and to experiment with various settings for the input parameters, we have developed three programs which correspond respectively to the deterministic version with a single tabu list (list T_1), the same with two tabu lists, and the probabilistic version.

To allow for extensive testing of the programs, we decided to use randomly generated problems. This has been a widely used practice in the past. In fact, for a number of years, most authors have tested their algorithms with problems generated by the uniform random graph scheme. This method works with two input parameters: a vertex-set-size **n** and a density **p**. It consists in examining each of the possible $\binom{n}{2}$ edges in the graph in sequence and adding them to the graph with probability p. Unfortunately, such graphs have been studied by researchers in random graph theory and it is now well-known that they possess very strong properties with regards to many attributes, one of them being the size of the maximum clique which most often falls in a very narrow range for a given choice of n and p (see [4, 5, 30, 31] for detailed results on this subject). To generate a more diverse sample of graphs, we thus decided to develop a new random graph generator, the $\hat{p}$-generator, which works with three rather than two inputs: **n** which is the vertex-set size as before, and **a** and **b**, two real numbers which satisfy $0 < a < b < 1$.

Procedure $\hat{p}$-generator (n, a, b)

begin

for i:=1 to n do $\hat{p}$[i]:=uniform (a, b);

for i:=1 to (n-1) do

for j:= (i+1) to n do

generate edge (i, j) with probability $\frac{\hat{p}[i] + \hat{p}[j]}{2}$

end.

Note that when a=b, one has $\hat{p}$[i]=a for all vertices and the $\hat{p}$-generator is then equivalent to the uniform random graph generator. The $\hat{p}$-generator can thus be seen as a generalization of the uniform generator.

The following properties apply in general to $\hat{p}$-generated graph:

(1) their expected density $\bar{p}$ is equal to (a+b)/2;

(2) for any vertex v_i, the conditional expected degree of v_i given $\hat{p}$[i] is $d(v_i|\hat{p}[i]) = \left(\frac{\hat{p}[i]}{2} + \frac{a+b}{4}\right)(n-1)$; which means that degree spread will increase rapidly with the difference (b-a);

(3) because of the larger degree spread, one would expect, for any fixed $\bar{p}$, the size of the maximum clique to increase with the difference (b-a); this intuitive assertion is confirmed by our empirical results.

A total of 240 test problems generated with the $\hat{p}$-generator were solved. For each of the four problem sizes, n=50, 100, 200, 400 vertices, 6 series of 10 problems corresponding to different settings of the a and b parameters were considered. This provided us with 6 families of random graphs: 3 of the classical type (when a=b) with expected densities of $\bar{p}$=0.25, 0.5, and 0.75, respectively, and 3 others of the new type (with $a \neq b$) with the same expected densities but presenting significant expected vertex degree spreads.

To allow for comparisons with other types of heuristics, the same problems were solved with two other methods. The first one is a straightforward implementation of Johnson's first-order greedy algorithm [25]. The other heuristic is a Simulated Annealing type procedure based on the "Rejectionless Method" of Greene and Supowit [20]. This method has been specially customized to deal with the MCP. In a parallel project on the application of Simulated Annealing to the MCP it was found to be the best performing of all tested methods [16]. All programs were implemented by the same programmer using the same language (PASCAL), compiler (TURBO PASCAL), and basic graph manipulation routines.

To experiment with the two tabu methods, we tested various settings of the input parameters. These ranged for the size of the T1 list from 0 to 150, for that of T2 from 0 to 40, and for MaxIter from 100 to 300. In the deterministic tabu, setting T2=0 (i.e. single-list tabu with T1 as the tabu list) provided very satisfactory results qualitywise, the best of which are presented in Table 2 under the ST (single tabu) column. On the other hand, with T1=0 (i.e. single list tabu with only T2), while execution times were significantly reduced, the quality of the solutions obtained was clearly inferior to that of T1>0. These observations also hold true for the probabilistic variant. Furthermore, for this second approach several ways of determining the values of parameters k and L were tested but no one method stood out as being superior. The only guidelines that could be inferred were that these parameters should take values big enough so as to promote a "wider" exploration of the solution space but not too much, thus preventing unnecessary "jumps" of the search process.

In Table 2, we have reported the average size of solutions for the best "Single-list Tabu" under ST and the best "Double-list Tabu" under DT. The other columns refer respectively to the first-order greedy heuristic

G1, the Rejectionless Method RM (average of 14 runs for each problem), the best Probabilistic Tabu PT, and the best known average solution for each 10 problem family BEST. In several cases, these "best" solutions have been determined to be optimal through the use of an exact branch-and-bound algorithm; they are identified by "*". Corresponding average execution times on an IBM PS/2 Model 70-121 with a 20 MHz 80386 CPU running under DOS are reported in Table 3. The parameter settings for the best tabu implementations are listed in Table 1. The same cooling schedule was used in the RM program for all 24 families of problems. It comprises 29 "temperature" levels, at each level t the "acceptance probability" is $p_t=(0.9)^t$ and the level length is $\ell_t = 200x(1.05)^{t-1}$. A more detailed description of this cooling schedule and the RM program can be found in [16].

We will now briefly comment on the results of Tables 2 and 3.

First, one can easily see that the average size of solutions for a fixed density $\bar{p}$ is always greater when a<b than when a=b, and that the difference between them increases with the spread (b-a). This confirms the intuitive assertion we made at the beginning of this section. Hence, the objective that was initially set forward when the $\hat{p}$-generator was conceived has effectively been attained.

The second comment one can make is that although G1 is much faster, it does not produce results whose quality is in line with that of the other methods. For their part, ST, DT, PT, and RM all produce very good quality results with very satisfactory execution times. It is interesting to note that in a fair number of cases the optimal solution was found by these methods.

In our experiments with parameter settings for the two tabu methods, we found out that the size of list T1 must not be too large and that list T2 must be fairly small (5 elements), otherwise the quality of the solutions starts to decrease. Interestingly enough, the "ideal" size of the lists seems to be independent of problem size, graph density and vertex degree spread. In some cases, it is possible to slightly improve the solutions by using a larger value of MaxIter, but the execution times are then much higher. It should be noted that since the probabilistic tabu performs sampling, it will run much faster on denser graphs ($\bar{p} > 0.5$). This makes it possible (and, to a certain extent, necessary) to use a larger value of MaxIter than in the deterministic variant.

With regards to comparisons with other methods reported in the literature, it should be stressed that our results were obtained on a micro-computer and that our CPU times cannot thus be compared directly with execution times on mainframe computers. It would certainly have been interesting to compare our approach with STABULUS, but unfortunately this was not possible.

7. Conclusion

We have presented two variants of a tabu search approach for the maximum clique problem. With proper parameter settings, these two variants have been shown to be able to produce very good solutions in a reasonable amount of CPU time on a micro-computer. When compared to other approximate methods for the MCP, our tabu algorithms were competitive with the best of those.

We have also proposed and implemented a new random graph generator which allows users to produce more varied graphs than classical techniques at just a slight increase in complexity and CPU time. This generator can be expected to become widely used for testing MCP algorithms in the coming years.

References

[1] G. AVONDO-BODENO, Economic Applications of the Theory of Graphs, Gordon & Breach Science Publishers, New York, 1962.

[2] E. BALAS and C.S. YU, Finding a maximum clique in an arbitrary graph, SIAM J. Computing 15 (1986), 1054-1068.

[3] C. BERGE, Théorie des graphes et ses applications, Dunod, Paris, 1962.

[4] B. BOLLOBÁS, Random Graphs, Academic Press, London, 1985.

[5] B. BOLLOBÁS and P. ERDÖS, Cliques in random graphs, Math. Proc. Camb. Phil. Soc. 80 (1976), 419-427.

[6] C. BRON and J. KERBOSCH, Finding all cliques of an undirected graph, Comm. ACM 16 (1973), 575-577.

[7] V. DEGOT and J.M. HUALDE, De l'utilisation de la notion de clique (sous-graphe complet symétrique) en matière de typologie des populations, Revue française d'automatique, informatique et recherche opérationnelle: Recherche opérationnelle 9 (1975), 5-18.

[8] N. DEO, Graph Theory with Applications to Engineering and Computer Science, Prentice-Hall, Englewood Cliffs, 1974.

[9] C. EBENEGGER, P.L. HAMMER, and D. DE WERRA, Pseudo-Boolean functions and stability of graphs, Annals of Discrete Mathematics 19 (1984), 83-98.

[10] C. FRIDEN, A. HERTZ, and D. DE WERRA, Stabulus: a technique for finding stable sets in large graphs with tabu search, Computing 42 (1989), 35-44.

[11] C. FRIDEN, A. HERTZ, and D. DE WERRA, Tabaris: An exact algorithm based on Tabu Search for finding a maximum independent set in a graph, Report ORWP 89/03, Département de Mathématiques, École Polytechnique Fédérale de Lausanne, February 1989.

[12] M.R. GAREY and D.S. JOHNSON, Computers and Intractibility: a Guide to the the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[13] M. GENDREAU, A fast greedy algorithm for the maximum clique problem, paper presented at the TIMS/ORSA Meeting, New Orleans, May 1987.

[14] M. GENDREAU, J.-C. PICARD, and L. ZUBIETA, An efficient implicit enumeration algorithm for the maximum clique problem, in Advances in Optimization and Control (H. A. Eiselt and G. Pederzoli, Eds.), Springer-Verlag, Berlin, 1988.

[15] M. GENDREAU, L. SALVAIL, and P. SORIANO, An appraisal of greedy heuristics for the maximum clique problem, Centre de recherche sur les transports, Université de Montréal, forthcoming.

[16] M. GENDREAU, P. SORIANO, and L. SALVAIL, Simulated annealing and cliques, Centre de recherche sur les transports, Université de Montréal, forthcoming; paper presented at the ORSA/TIMS Meeting, New York, October 1989.

[17] L. GERHARDS and W. LINDENBERG, Clique detection for nondirected graphs: two new algorithms, Computing 21 (1979), 295-549.

[18] F. GLOVER, Future paths for integer programming and links to artificial intelligence, Computers and O.R. 13 (1986), 533-549.

[19] F. GLOVER, Tabu Search Part I, ORSA Journal on Computing 1 (1989), 190-206.

[20] J.W. GREENE and K.J. SUPOWIT, Simulated annealing without rejected moves, IEEE Trans. Computer Aided Design CAD-5 (1986), 221-228.

[21] M. GRÖTSCHEL, L. LOVASZ, and A. SCHRIJVER, Geometric Algorithms and Combinatorial Optimization, Springer-Verlag, Berlin, 1988.

[22] P. HANSEN and B. JAUMARD, Algorithms for the maximum satisfiability problem, RUTCOR Research Report 43-87, Rutgers University, 1987.

[23] F. HARARY, Graph theory as a structural model in the social sciences, in Graph Theory and its Applications (B. Harris, ed.), Academic Press, New York, 1970.

[24] A. HERTZ and D. DE WERRA, Using Tabu Search techniques for graph coloring, Computing 29 (1987), 345-351.

[25] D.S. JOHNSON, Approximation algorithms for combinatorial problems, Journal of Computer and Systems Sciences 9 (1974), 256-278.

[26] D.S. JOHNSON, M. YANNAKAKIS, and C.H. PAPADIMITRIOU, On generating all maximal independent sets, Information Processing Letters 27 (1988), 119-123.

[27] S. KIRKPATRICK, C.D. GELATT Jr., and M.P. VECCHI, Optimization by simulated annealing, Science 220 (1983), 671-680.

[28] E. LOUKAKIS, A new backtracking algorithm for generating the family of maximal independent sets of a graph, Computer and Mathematics with Applications 9 (1983), 583-589.

[29] E. LOUKAKIS and C. TSOUROS, Determining the number of internal stability of a graph, International J. Computer Math 11 (1982), 207-220.

[30] D.W. MATULA, The employee party problem, Not. A.M.S. 19 (1972), A-382.

[31] D.W. MATULA, The largest clique in a random graph, Technical Report CS7608, Southern Methodist University, 1976.

[32] P.M. PARDALOS and G.P. RODGERS, A branch and bound algorithm for the maximum clique problem, to appear in Mathematical Programming.

[33] B. ROY, Algèbre moderne et théorie des graphes, Tome 1, Dunod, Paris, 1969.

[34] C.E. SHANNON, The zero-error capacity of a noisy channel, Composium on Information Theory, I.R.E. Transactions 3 (1956).

[35] R.E. TARJAN and A.E. TROJANOWSKI, Finding a maximum independent set, SIAM J. on Computing 6 (1977), 537-546.

[36] J. TURNER and W.H. KAUTZ, A survey of progress in graph theory in the Soviet Union, SIAM 12 (1970).

[37] D. DE WERRA and A. HERTZ, Tabu Search techniques: A tutorial and application to neural networks, OR Spektrum 11 (1989), 131-141.

[38] M. WIDMER and A. HERTZ, A new heuristic method for solving the flow shop sequencing problem, European Journal of Operational Research 41 (1989), 186-193.

PARAMETER	ST	DT	PT
T1	100	100	150
T2	---	5	5
MaxIter	250	250	300

Table 1 : Parameter settings

n	$\bar{p}$	[a,b]	G1	RM	ST	DT	PT	BEST
50	0.25	0.25 - 0.25	4.4	4.8	4.8	4.8	4.8	4.8*
		0.0 - 0.5	4.5	5.1	5.1	5.1	5.1	5.1*
	0.5	0.5 - 0.5	7.4	7.8	7.8	7.7	7.8	7.8*
		0.0 - 1.0	9.7	9.9	9.9	9.9	9.9	9.9*
	0.75	0.75 - 0.75	12.4	13.2	13.2	13.2	13.1	13.2*
		0.5 - 1.0	15.2	15.4	15.3	15.4	15.3	15.4*
100	0.25	0.25 - 0.25	4.9	5.3	5.3	5.3	5.3	5.3*
		0.0 - 0.5	5.7	6.3	6.3	6.3	6.3	6.3*
	0.5	0.5 - 0.5	8.6	9.3	9.2	9.3	9.3	9.3*
		0.0 - 1.0	14.0	14.8	14.8	14.8	14.8	14.8*
	0.75	0.75 - 0.75	15.4	16.5	16.4	16.3	16.4	16.6
		0.5 - 1.0	20.9	21.7	21.8	21.7	21.6	21.8
200	0.25	0.25 - 0.25	5.3	6.2	6.1	6.2	6.1	6.2*
		0.0 - 0.5	6.6	7.5	7.4	7.4	7.5	7.5*
	0.5	0.5 - 0.5	9.7	10.9	10.3	10.7	10.4	11.1
		0.0 - 1.0	19.8	21.3	21.0	21.1	21.1	21.3
	0.75	0.75 - 0.75	18.5	20.3	19.8	19.9	20.0	20.5
		0.5 - 1.0	29.8	31.0	30.9	31.0	30.8	31.2
400	0.25	0.25 - 0.25	6.1	7.0	6.9	7.0	6.9	7.2
		0.0 - 0.5	7.8	8.7	8.5	8.7	8.5	8.8
	0.5	0.5 - 0.5	10.9	12.0	11.9	11.8	11.8	12.1
		0.0 - 1.0	29.3	31.1	31.1	31.1	31.2	31.3
	0.75	0.75 - 0.75	21.9	23.2	22.9	23.3	23.4	23.5
		0.5 - 1.0	40.5	43.0	43.0	43.1	43.1	43.2

Table 2 : Average size of solutions

n	$\bar{p}$	[a,b]	G1	RM	ST	DT	PT
50	0.25	0.25 - 0.25	0.00	24.76	10.33	9.51	8.79
		0.0 - 0.5	0.00	25.10	9.38	8.66	8.67
	0.5	0.5 - 0.5	0.00	27.54	8.60	7.33	10.99
		0.0 - 1.0	0.00	29.69	9.37	7.95	7.91
	0.75	0.75 - 0.75	0.00	32.00	13.27	11.20	10.62
		0.5 - 1.0	0.05	33.87	14.38	13.39	8.77
100	0.25	0.25 - 0.25	0.00	27.62	10.01	9.62	15.23
		0.0 - 0.5	0.00	28.55	10.40	9.95	14.47
	0.5	0.5 - 0.5	0.03	32.38	11.20	10.25	15.28
		0.0 - 1.0	0.10	39.96	16.99	14.76	11.62
	0.75	0.75 - 0.75	0.10	41.55	21.42	19.64	15.54
		0.5 - 1.0	0.17	48.27	29.35	25.45	14.79
200	0.25	0.25 - 0.25	0.10	32.56	12.85	13.08	22.54
		0.0 - 0.5	0.10	34.51	12.52	13.49	21.66
	0.5	0.5 - 0.5	0.20	40.62	14.33	15.32	21.79
		0.0 - 1.0	0.41	62.00	35.23	33.54	21.49
	0.75	0.75 - 0.75	0.46	59.49	39.02	34.70	37.15
		0.5 - 1.0	0.81	81.02	64.70	65.05	33.13
400	0.25	0.25 - 0.25	0.39	42.34	17.83	18.38	29.78
		0.0 - 0.5	0.43	46.35	17.61	18.88	27.08
	0.5	0.5 - 0.5	0.70	56.65	26.49	22.81	30.04
		0.0 - 1.0	1.90	117.03	98.80	39.85	54.74
	0.75	0.75 - 0.75	1.64	91.96	64.45	64.49	79.53
		0.5 - 1.0	3.28	155.30	181.50	178.46	106.00

Table 3 : Average execution time (CPU Seconds)